CSCI 3901

Software Development Concepts

Assignment 4

Team Members

B00874162 - Vivekkumar Rameshbhai Patel

Boggle Puzzle:

Assignment 4 Overview:

- For boggle game there a is grid of letters is given and task is to find all possible word for which there is a path possible. While traversing in grid traversal can be made in the any direction (left , right, up, down and diagonally).While finding the word if any letter of the word is visited then it can't be used again to find the same word. Moreover, while finding word only those letters are visited which are adjacent to the current letter of the word.

Strategy to solve Boggle Game:

- To find word that exist in the puzzle grid or not I have used recursive function to visit each direction of the letter but I have improved execution time of the program by not executing function for each and every pair of coordinates of the letter if letter appears more than once in the puzzle grid.
- Before commencing searching process for the each word I have stored pair of coordinates of each element of the puzzle grid. Hence, it is faster to locate starting position of the word in puzzle grid.
- To find starting position of the word priority is given to those pair of coordinates, having minimum X and minimum Y coordinate. If word found for such pair of coordinates then searching will not be done for further pair of coordinates.

Algorithm to solve Boggle Game:

1) Find x and y coordinate of each element of the **NxN** puzzle grid and store it in the hash map(**Key=element of the grid**, **Value=x,y**).(Value will be string separating x and y coordinate by ,)
2) Sort the list of coordinates for each element of the grid if the element appear more than once in the grid. Keep pair of X and Y coordinates on the

top whose X and Y coordinates is minimal than list of coordinates for the same element.

3) Iterate through list of words of the dictionary and do following for each word.
    3.1 Get the coordinate for initial letter of the word to locate starting position in the grid for the word.
    3.2 Mark the starting position as visited.
    3.3 Visit each 8 adjacent direction of the letter to find next letter of the word as per the sequence of letter in the word.
    3.4 If a letter found in either of the direction as per the sequence of letters in word then keep it as visited and move to that direction and a make a new letter current letter of the grid and again repeat from step 3.3 for another eight adjacent direction of the current letter.
    3.5 If word is found then add it into list of word that can be found in the puzzle and skip checking for other list of coordinates of the same initial letter of the word as word has been found for the minimum X and Y coordinate of the word else repeat from step 3.1 to 3.4 for other list of coordinates of the same initial letter of the word.
    3.6 If word is not found for either of the list of coordinate of the word then repeat steps 3.1 to 3.5 for other words.

Degree of efficiency:

- I have used Hash Map data structure to store coordinates of each element of the puzzle grid to obtain **O (1)** time complexity while searching the position of the initial letter of each word instead traversing entire 2D array .
- For the traversal of the 2D array I have used concept of **DFS** using recursion. According to that I traverse as many as direction possible before the backtracking.
- **In order to do not explore each trivial grids,** If any letter appear more than once in the puzzle grid then coordinates of those each cell of the letter is stored in the hash map with letter as key and each pair of coordinates as a list. This list of coordinates are sorted in order to satisfies the constraints of reporting a word with minimum Y coordinates if there is tie in between X coordinates of the word else report a word with minimum X coordinates.

Hence, a pair of coordinates with minimum Y coordinates will be on the top of the list.

- While, searching a word, searching process will be done first for the pair of coordinates which are on the top of the list and if word found then searching process will not be done for other list of the coordinates. Hence, it will explore only **non-trivial grids**. Which reduce the time of execution for the recursive function by not executing for each possibilities of the word and increase the efficiency of the program.

Files and external data:

There are total 2 files:

- **BoggleInterface.java:** It contains methods that needs to be implemented for implementation of **Boggle**.

-**Boggle.java:** Class that contains implementation of the methods available in **BoggleInterface** to find words of the dictionary from the puzzle grid.

Data structures and their relations to each other:

Data structured used in Boggle class

Boggle.java:

Map<String, List<String>> coordinateMapping: It is hash map used to store coordinates of each elements of the puzzle grid. In which name of elements is used as a key and list of coordinates for the element is used as a value.

-**List<String> dictionary:** It stores list of word of the given dictionary. Which needs to be find from the given puzzle grid.

-**List<String> lettersOfGrid**: It is list which stores elements of the puzzle grid while reading from the buffer stream. Each line of the puzzle grid is considered as single string.

-**String board[][]:**

Once data is read from the file for the puzzle grid 2D array of puzzle board is created from the data stored in **lettersOfGrid**. This puzzle board is used to search each word of the dictionary word.

Test Cases:

- Stream is null while reading dictionary of the word.
- File is empty while reading dictionary of the words.
- Dictionary contains words having less than 2 characters.
- Blank line encounter while reading words of the dictionary.
- Stream is null while reading data for the grid of the puzzle.
- File is empty while reading grid of the puzzle.
- Blank line encounter while reading data of the grid.
- Number of characters in each row of the grid are unequal.
- Solve puzzle for 0x0 puzzle grid.
- Solve puzzle for the grid having unequal number of characters in row.
- Solve puzzle if grid is valid and dictionary of the word is empty.
- Solve puzzle if grid is valid and dictionary contains only one word
- Solve puzzle if grid is valid and dictionary contains more than one words.
- print puzzle if puzzle grid is empty.
- print puzzle grid if puzzle grid is invalid.
- Call print method before calling getPuzzle() and getDictionary() method.
- Call to solve() method before getPuzzle() and getDictionary() methods.